FOUNDATION-0.2c: Workflow History Tables - PART 1 (Code)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 1 Afternoon)
Component: Workflow Execution History & Tracking
Estimated Time: 15 min AI execution + 15 min verification
Complexity: MEDIUM
Risk Level: LOW
Files: Part 1 of 2 (Code implementation)

📦 DEPENDENCIES

Must Complete First:

- **FOUNDATION-0.2a:** Core Database Schema ✅ COMPLETED
  - All 6 core tables created
  - 13 tests passing
- **FOUNDATION-0.2b:** Agent State Tables ✅ COMPLETED
  - All 4 agent state tables created
  - 16 tests passing
  - Total: 29/29 tests passing

Required Services Running:

bash

```
# Verify all services are healthy
cd ~/optiinfra
make verify

# Expected output:
# PostgreSQL... ✅ HEALTHY
# ClickHouse... ✅ HEALTHY
# Qdrant... ✅ HEALTHY
# Redis... ✅ HEALTHY
```

Database State:

bash

```
# Verify existing tables
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "\dt"

# Expected: 10 tables from 0.2a + 0.2b
# customers, agents, events, recommendations, approvals, optimizations
# agent_configs, agent_states, agent_capabilities, agent_metrics
```

🎯 OBJECTIVE

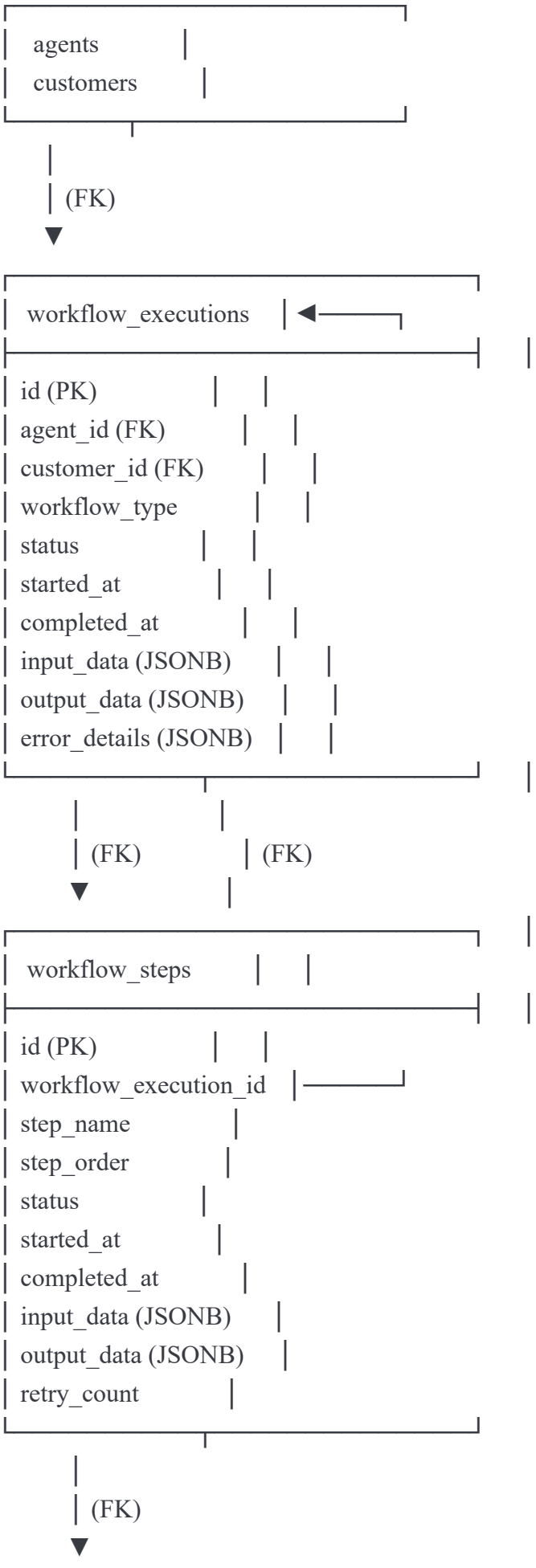
Add **workflow execution tracking tables** to enable complete audit trails and workflow management. These tables track:

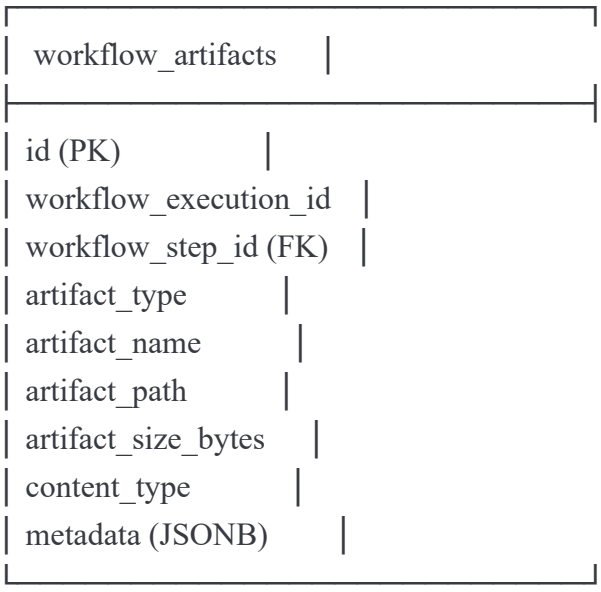
1. **Workflow Executions** - High-level workflow runs (cost analysis, performance tuning, etc.)
2. **Workflow Steps** - Individual steps within each workflow
3. **Workflow Artifacts** - Files and outputs generated during workflows

What This Enables:

- ✅ Track all workflow executions with start/end times
- ✅ Monitor step-by-step progress in real-time
- ✅ Debug failed workflows (see which step failed and why)
- ✅ Store generated artifacts (reports, configs, recommendations)
- ✅ Link workflows to agents, customers, and recommendations
- ✅ Enable workflow recovery and retry logic
- ✅ Complete audit trail for compliance

Database Design:


agents
customers
(FK)
workflow_executions
id (PK)
agent_id (FK)
customer_id (FK)
workflow_type
status
started_at
completed_at
input_data (JSONB)
output_data (JSONB)
error_details (JSONB)
(FK)
(FK)
workflow_steps
id (PK)
workflow_execution_id
step_name
step_order
status
started_at
completed_at
input_data (JSONB)
output_data (JSONB)
retry_count
(FK)



📁 FILE 1: Workflow History Models

Location: `~/optiinfra/shared/database/models/workflow_history.py`

python

```
"""
Workflow History Models (FOUNDATION-0.2c)
Tracks workflow executions, steps, and artifacts
"""

from datetime import datetime
from sqlalchemy import (
    Column, String, DateTime, Integer, BigInteger, Text, Enum as SQLEnum,
    ForeignKey, Index
)
from sqlalchemy.dialects.postgresql import UUID, JSONB
from sqlalchemy.orm import relationship
import uuid
import enum

from shared.database.base import Base


# ============================================
# ENUMS
# ============================================

class WorkflowType(str, enum.Enum):
    """Types of workflows that can be executed"""
    COST_ANALYSIS = "cost_analysis"
    PERFORMANCE_TUNING = "performance_tuning"
    RESOURCE_OPTIMIZATION = "resource_optimization"
    QUALITY_CHECK = "quality_check"
    SCALING_DECISION = "scaling_decision"
    CONFIGURATION_UPDATE = "configuration_update"
    HEALTH_CHECK = "health_check"
    ANOMALY_DETECTION = "anomaly_detection"


class WorkflowStatus(str, enum.Enum):
    """Status of workflow execution"""
    PENDING = "pending"
    RUNNING = "running"
    COMPLETED = "completed"
    FAILED = "failed"
    CANCELLED = "cancelled"
```

```
    TIMEOUT = "timeout"


class StepStatus(str, enum.Enum):
    """Status of individual workflow step"""
    PENDING = "pending"
    RUNNING = "running"
    COMPLETED = "completed"
    FAILED = "failed"
    SKIPPED = "skipped"
    RETRYING = "retrying"


class ArtifactType(str, enum.Enum):
    """Types of artifacts generated during workflows"""
    REPORT = "report"
    CONFIG = "config"
    LOG = "log"
    RECOMMENDATION = "recommendation"
    CHART = "chart"
    METRICS = "metrics"
    ALERT = "alert"
    DIAGNOSTIC = "diagnostic"


# ============================================
# MODELS
# ============================================

class WorkflowExecution(Base):
    """
    Tracks high-level workflow executions

    A workflow execution represents a complete run of a workflow
    (e.g., a cost analysis, performance tuning session)
    """
    __tablename__ = "workflow_executions"

    # Primary Key
    id = Column(
        UUID(as_uuid=True),
```

```
    primary_key=True,
    default=uuid.uuid4,
    index=True
)

# Foreign Keys
agent_id = Column(
    UUID(as_uuid=True),
    ForeignKey("agents.id", ondelete="CASCADE"),
    nullable=False,
    index=True
)
customer_id = Column(
    UUID(as_uuid=True),
    ForeignKey("customers.id", ondelete="CASCADE"),
    nullable=False,
    index=True
)

# Workflow Details
workflow_type = Column(
    SQLEnum(WorkflowType, name="workflow_type"),
    nullable=False,
    index=True
)
status = Column(
    SQLEnum(WorkflowStatus, name="workflow_status"),
    nullable=False,
    default=WorkflowStatus.PENDING,
    index=True
)

# Timing
started_at = Column(DateTime, nullable=True, index=True)
completed_at = Column(DateTime, nullable=True)

# Data
input_data = Column(JSONB, nullable=False, default=dict)
output_data = Column(JSONB, nullable=False, default=dict)
error_details = Column(JSONB, nullable=True)
```

```
# Metadata
metadata = Column(JSONB, nullable=False, default=dict)

# Timestamps
created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
updated_at = Column(
    DateTime,
    default=datetime.utcnow,
    onupdate=datetime.utcnow,
    nullable=False
)

# Relationships
agent = relationship("Agent", back_populates="workflow_executions")
customer = relationship("Customer", back_populates="workflow_executions")
steps = relationship(
    "WorkflowStep",
    back_populates="execution",
    cascade="all, delete-orphan",
    order_by="WorkflowStep.step_order"
)
artifacts = relationship(
    "WorkflowArtifact",
    back_populates="execution",
    cascade="all, delete-orphan"
)

# Indexes
__table_args__ = (
    Index(
        "ix_workflow_executions_agent_status",
        "agent_id", "status"
    ),
    Index(
        "ix_workflow_executions_customer_type",
        "customer_id", "workflow_type"
    ),
    Index(
        "ix_workflow_executions_started_at",
        "started_at"
    ),
```

```
    )

    def __repr__(self):
        return (
            f"<WorkflowExecution(id={self.id}, "
            f"type={self.workflow_type}, status={self.status})>"
        )


class WorkflowStep(Base):
    """
    Tracks individual steps within a workflow execution

    Each workflow consists of multiple steps that execute in order
    """
    __tablename__ = "workflow_steps"

    # Primary Key
    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        index=True
    )

    # Foreign Key
    workflow_execution_id = Column(
        UUID(as_uuid=True),
        ForeignKey("workflow_executions.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )

    # Step Details
    step_name = Column(String(255), nullable=False, index=True)
    step_order = Column(Integer, nullable=False)
    status = Column(
        SQLEnum(StepStatus, name="step_status"),
        nullable=False,
        default=StepStatus.PENDING,
        index=True
```

```
)

# Timing
started_at = Column(DateTime, nullable=True)
completed_at = Column(DateTime, nullable=True)

# Data
input_data = Column(JSONB, nullable=False, default=dict)
output_data = Column(JSONB, nullable=False, default=dict)
error_details = Column(JSONB, nullable=True)

# Retry Logic
retry_count = Column(Integer, nullable=False, default=0)
max_retries = Column(Integer, nullable=False, default=3)

# Metadata
metadata = Column(JSONB, nullable=False, default=dict)

# Timestamps
created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
updated_at = Column(
    DateTime,
    default=datetime.utcnow,
    onupdate=datetime.utcnow,
    nullable=False
)

# Relationships
execution = relationship("WorkflowExecution", back_populates="steps")
artifacts = relationship(
    "WorkflowArtifact",
    back_populates="step",
    cascade="all, delete-orphan"
)

# Indexes
__table_args__ = (
    Index(
        "ix_workflow_steps_execution_order",
        "workflow_execution_id", "step_order"
    ),
```

```
        Index(
            "ix_workflow_steps_execution_status",
            "workflow_execution_id", "status"
        ),
        Index(
            "ix_workflow_steps_name_status",
            "step_name", "status"
        ),
    )

    def __repr__(self):
        return (
            f"<WorkflowStep(id={self.id}, name={self.step_name}, "
            f"order={self.step_order}, status={self.status})>"
        )


class WorkflowArtifact(Base):
    """
    Stores artifacts generated during workflow execution

    Artifacts include reports, configs, logs, charts, etc.
    """
    __tablename__ = "workflow_artifacts"

    # Primary Key
    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        index=True
    )

    # Foreign Keys
    workflow_execution_id = Column(
        UUID(as_uuid=True),
        ForeignKey("workflow_executions.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
    workflow_step_id = Column(
```

```
    UUID(as_uuid=True),
    ForeignKey("workflow_steps.id", ondelete="CASCADE"),
    nullable=True,  # Can be null for execution-level artifacts
    index=True
)

# Artifact Details
artifact_type = Column(
    SQLEnum(ArtifactType, name="artifact_type"),
    nullable=False,
    index=True
)
artifact_name = Column(String(255), nullable=False)
artifact_path = Column(Text, nullable=False)  # S3 path, file path, etc.

# Size & Type
artifact_size_bytes = Column(BigInteger, nullable=True)
content_type = Column(String(100), nullable=True)  # MIME type

# Metadata
metadata = Column(JSONB, nullable=False, default=dict)

# Timestamps
created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
updated_at = Column(
    DateTime,
    default=datetime.utcnow,
    onupdate=datetime.utcnow,
    nullable=False
)

# Relationships
execution = relationship("WorkflowExecution", back_populates="artifacts")
step = relationship("WorkflowStep", back_populates="artifacts")

# Indexes
__table_args__ = (
    Index(
        "ix_workflow_artifacts_execution_type",
        "workflow_execution_id", "artifact_type"
    ),
```

```
        Index(
            "ix_workflow_artifacts_step_type",
            "workflow_step_id", "artifact_type"
        ),
    )

    def __repr__(self):
        return (
            f"<WorkflowArtifact(id={self.id}, name={self.artifact_name}, "
            f"type={self.artifact_type})>"
        )
```

📁 FILE 2: Update Core Models (Add Relationships)

Location: `~/optiinfra/shared/database/models/core.py`

Add these relationships to existing models:

python

```python
# In the Agent model, add:
from sqlalchemy.orm import relationship

class Agent(Base):
    # ... existing fields ...

    # ADD THIS NEW RELATIONSHIP:
    workflow_executions = relationship(
        "WorkflowExecution",
        back_populates="agent",
        cascade="all, delete-orphan"
    )


# In the Customer model, add:
class Customer(Base):
    # ... existing fields ...

    # ADD THIS NEW RELATIONSHIP:
    workflow_executions = relationship(
        "WorkflowExecution",
        back_populates="customer",
        cascade="all, delete-orphan"
    )


# In the AgentState model (from 0.2b), add:
class AgentState(Base):
    # ... existing fields ...

    # ADD THIS NEW FIELD (optional - for linking current workflow):
    current_workflow_id = Column(
        UUID(as_uuid=True),
        ForeignKey("workflow_executions.id", ondelete="SET NULL"),
        nullable=True,
        index=True
    )

    # ADD THIS NEW RELATIONSHIP:
    current_workflow = relationship(
        "WorkflowExecution",
```

```
    foreign_keys=[current_workflow_id]
)
```

📁 FILE 3: Update Model Imports

Location: `~/optiinfra/shared/database/models/__init__.py`

```python
```

```
"""
Database models package
"""

# Core models (0.2a)
from shared.database.models.core import (
    Customer,
    Agent,
    Event,
    Recommendation,
    Approval,
    Optimization,
    # Enums
    CustomerStatus,
    AgentType,
    AgentStatus,
    EventType,
    EventSeverity,
    RecommendationType,
    ApprovalStatus,
    OptimizationStatus,
)

# Agent state models (0.2b)
from shared.database.models.agent_state import (
    AgentConfig,
    AgentState,
    AgentCapability,
    AgentMetric,
    # Enums
    ConfigType,
    AgentStatusDetail,
    MetricType,
)

# Workflow history models (0.2c) - NEW!
from shared.database.models.workflow_history import (
    WorkflowExecution,
    WorkflowStep,
    WorkflowArtifact,
    # Enums
```

```
    WorkflowType,
    WorkflowStatus,
    StepStatus,
    ArtifactType,
)

__all__ = [
    # Core models
    "Customer",
    "Agent",
    "Event",
    "Recommendation",
    "Approval",
    "Optimization",
    # Core enums
    "CustomerStatus",
    "AgentType",
    "AgentStatus",
    "EventType",
    "EventSeverity",
    "RecommendationType",
    "ApprovalStatus",
    "OptimizationStatus",
    # Agent state models
    "AgentConfig",
    "AgentState",
    "AgentCapability",
    "AgentMetric",
    # Agent state enums
    "ConfigType",
    "AgentStatusDetail",
    "MetricType",
    # Workflow history models - NEW!
    "WorkflowExecution",
    "WorkflowStep",
    "WorkflowArtifact",
    # Workflow history enums - NEW!
    "WorkflowType",
    "WorkflowStatus",
    "StepStatus",
```

```
    "ArtifactType",
]
```

📁 FILE 4: Alembic Migration

Location: `~/optiinfra/shared/database/migrations/versions/003_workflow_history.py`

python

```
"""Create workflow history tables

Revision ID: 003_workflow_history
Revises: 002_agent_state_tables
Create Date: 2025-01-19 14:00:00.000000

"""
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.
revision = '003_workflow_history'
down_revision = '002_agent_state_tables'
branch_labels = None
depends_on = None


def upgrade() -> None:
    """Create workflow history tables"""

    # Create ENUM types
    workflow_type_enum = postgresql.ENUM(
        'cost_analysis',
        'performance_tuning',
        'resource_optimization',
        'quality_check',
        'scaling_decision',
        'configuration_update',
        'health_check',
        'anomaly_detection',
        name='workflow_type',
        create_type=True
    )
    workflow_type_enum.create(op.get_bind(), checkfirst=True)

    workflow_status_enum = postgresql.ENUM(
        'pending',
        'running',
        'completed',
        'failed',
```

```
    'cancelled',
    'timeout',
    name='workflow_status',
    create_type=True
)
workflow_status_enum.create(op.get_bind(), checkfirst=True)

step_status_enum = postgresql.ENUM(
    'pending',
    'running',
    'completed',
    'failed',
    'skipped',
    'retrying',
    name='step_status',
    create_type=True
)
step_status_enum.create(op.get_bind(), checkfirst=True)

artifact_type_enum = postgresql.ENUM(
    'report',
    'config',
    'log',
    'recommendation',
    'chart',
    'metrics',
    'alert',
    'diagnostic',
    name='artifact_type',
    create_type=True
)
artifact_type_enum.create(op.get_bind(), checkfirst=True)

# Create workflow_executions table
op.create_table(
    'workflow_executions',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('customer_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('workflow_type', workflow_type_enum, nullable=False),
    sa.Column('status', workflow_status_enum, nullable=False),
```

```
    sa.Column('started_at', sa.DateTime(), nullable=True),
    sa.Column('completed_at', sa.DateTime(), nullable=True),
    sa.Column('input_data', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('output_data', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('error_details', postgresql.JSONB(), nullable=True),
    sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.Column('updated_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['agent_id'], ['agents.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['customer_id'], ['customers.id'], ondelete='CASCADE'),
)

# Create indexes for workflow_executions
op.create_index('ix_workflow_executions_id', 'workflow_executions', ['id'])
op.create_index('ix_workflow_executions_agent_id', 'workflow_executions', ['agent_id'])
op.create_index('ix_workflow_executions_customer_id', 'workflow_executions', ['customer_id'])
op.create_index('ix_workflow_executions_workflow_type', 'workflow_executions', ['workflow_type'])
op.create_index('ix_workflow_executions_status', 'workflow_executions', ['status'])
op.create_index('ix_workflow_executions_started_at', 'workflow_executions', ['started_at'])
op.create_index('ix_workflow_executions_agent_status', 'workflow_executions', ['agent_id', 'status'])
op.create_index('ix_workflow_executions_customer_type', 'workflow_executions', ['customer_id', 'workflow_type'])

# Create workflow_steps table
op.create_table(
    'workflow_steps',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('workflow_execution_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('step_name', sa.String(255), nullable=False),
    sa.Column('step_order', sa.Integer(), nullable=False),
    sa.Column('status', step_status_enum, nullable=False),
    sa.Column('started_at', sa.DateTime(), nullable=True),
    sa.Column('completed_at', sa.DateTime(), nullable=True),
    sa.Column('input_data', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('output_data', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('error_details', postgresql.JSONB(), nullable=True),
    sa.Column('retry_count', sa.Integer(), nullable=False, server_default='0'),
    sa.Column('max_retries', sa.Integer(), nullable=False, server_default='3'),
    sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.Column('updated_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['workflow_execution_id'], ['workflow_executions.id'], ondelete='CASCADE'),
```

```
)

# Create indexes for workflow_steps
op.create_index('ix_workflow_steps_id', 'workflow_steps', ['id'])
op.create_index('ix_workflow_steps_workflow_execution_id', 'workflow_steps', ['workflow_execution_id'])
op.create_index('ix_workflow_steps_step_name', 'workflow_steps', ['step_name'])
op.create_index('ix_workflow_steps_status', 'workflow_steps', ['status'])
op.create_index('ix_workflow_steps_execution_order', 'workflow_steps', ['workflow_execution_id', 'step_order'])
op.create_index('ix_workflow_steps_execution_status', 'workflow_steps', ['workflow_execution_id', 'status'])
op.create_index('ix_workflow_steps_name_status', 'workflow_steps', ['step_name', 'status'])

# Create workflow_artifacts table
op.create_table(
    'workflow_artifacts',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('workflow_execution_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('workflow_step_id', postgresql.UUID(as_uuid=True), nullable=True),
    sa.Column('artifact_type', artifact_type_enum, nullable=False),
    sa.Column('artifact_name', sa.String(255), nullable=False),
    sa.Column('artifact_path', sa.Text(), nullable=False),
    sa.Column('artifact_size_bytes', sa.BigInteger(), nullable=True),
    sa.Column('content_type', sa.String(100), nullable=True),
    sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.Column('updated_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['workflow_execution_id'], ['workflow_executions.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['workflow_step_id'], ['workflow_steps.id'], ondelete='CASCADE'),
)

# Create indexes for workflow_artifacts
op.create_index('ix_workflow_artifacts_id', 'workflow_artifacts', ['id'])
op.create_index('ix_workflow_artifacts_workflow_execution_id', 'workflow_artifacts', ['workflow_execution_id'])
op.create_index('ix_workflow_artifacts_workflow_step_id', 'workflow_artifacts', ['workflow_step_id'])
op.create_index('ix_workflow_artifacts_artifact_type', 'workflow_artifacts', ['artifact_type'])
op.create_index('ix_workflow_artifacts_execution_type', 'workflow_artifacts', ['workflow_execution_id', 'artifact_type']
op.create_index('ix_workflow_artifacts_step_type', 'workflow_artifacts', ['workflow_step_id', 'artifact_type'])

# Add current_workflow_id to agent_states (optional link)
op.add_column(
    'agent_states',
    sa.Column('current_workflow_id', postgresql.UUID(as_uuid=True), nullable=True)
```

```
    )
    op.create_foreign_key(
        'fk_agent_states_current_workflow',
        'agent_states',
        'workflow_executions',
        ['current_workflow_id'],
        ['id'],
        ondelete='SET NULL'
    )
    op.create_index('ix_agent_states_current_workflow_id', 'agent_states', ['current_workflow_id'])


def downgrade() -> None:
    """Drop workflow history tables"""

    # Drop indexes and column from agent_states
    op.drop_index('ix_agent_states_current_workflow_id', 'agent_states')
    op.drop_constraint('fk_agent_states_current_workflow', 'agent_states', type_='foreignkey')
    op.drop_column('agent_states', 'current_workflow_id')

    # Drop workflow_artifacts table and indexes
    op.drop_index('ix_workflow_artifacts_step_type', 'workflow_artifacts')
    op.drop_index('ix_workflow_artifacts_execution_type', 'workflow_artifacts')
    op.drop_index('ix_workflow_artifacts_artifact_type', 'workflow_artifacts')
    op.drop_index('ix_workflow_artifacts_workflow_step_id', 'workflow_artifacts')
    op.drop_index('ix_workflow_artifacts_workflow_execution_id', 'workflow_artifacts')
    op.drop_index('ix_workflow_artifacts_id', 'workflow_artifacts')
    op.drop_table('workflow_artifacts')

    # Drop workflow_steps table and indexes
    op.drop_index('ix_workflow_steps_name_status', 'workflow_steps')
    op.drop_index('ix_workflow_steps_execution_status', 'workflow_steps')
    op.drop_index('ix_workflow_steps_execution_order', 'workflow_steps')
    op.drop_index('ix_workflow_steps_status', 'workflow_steps')
    op.drop_index('ix_workflow_steps_step_name', 'workflow_steps')
    op.drop_index('ix_workflow_steps_workflow_execution_id', 'workflow_steps')
    op.drop_index('ix_workflow_steps_id', 'workflow_steps')
    op.drop_table('workflow_steps')

    # Drop workflow_executions table and indexes
    op.drop_index('ix_workflow_executions_customer_type', 'workflow_executions')
```

```
    op.drop_index('ix_workflow_executions_agent_status', 'workflow_executions')
    op.drop_index('ix_workflow_executions_started_at', 'workflow_executions')
    op.drop_index('ix_workflow_executions_status', 'workflow_executions')
    op.drop_index('ix_workflow_executions_workflow_type', 'workflow_executions')
    op.drop_index('ix_workflow_executions_customer_id', 'workflow_executions')
    op.drop_index('ix_workflow_executions_agent_id', 'workflow_executions')
    op.drop_index('ix_workflow_executions_id', 'workflow_executions')
    op.drop_table('workflow_executions')

    # Drop ENUM types
    sa.Enum(name='artifact_type').drop(op.get_bind(), checkfirst=True)
    sa.Enum(name='step_status').drop(op.get_bind(), checkfirst=True)
    sa.Enum(name='workflow_status').drop(op.get_bind(), checkfirst=True)
    sa.Enum(name='workflow_type').drop(op.get_bind(), checkfirst=True)
```

🚀 EXECUTION STEPS

Step 1: Create Models File

bash

```
cd ~/optiinfra/shared/database/models

# Create workflow_history.py
cat > workflow_history.py << 'EOF'
[Copy the entire workflow_history.py content from FILE 1 above]
EOF

# Verify file created
ls -lh workflow_history.py
# Expected: ~400 lines
```

Step 2: Update Core Models

bash

```
cd ~/optiinfra/shared/database/models

# Open core.py and add relationships
nano core.py

# Add to Agent class:
#   workflow_executions = relationship(...)
#
# Add to Customer class:
#   workflow_executions = relationship(...)

# Or use sed to append (careful!):
# Add workflow_executions relationship to Agent model
# Add workflow_executions relationship to Customer model
```

Step 3: Update agent_state.py

bash

```
cd ~/optiinfra/shared/database/models

# Open agent_state.py and update AgentState model
nano agent_state.py

# Add to AgentState class:
#   current_workflow_id = Column(...)
#   current_workflow = relationship(...)
```

Step 4: Update Model Imports

bash

```
cd ~/optiinfra/shared/database/models

# Update __init__.py with workflow models
cat > __init__.py << 'EOF'
[Copy the entire __init__.py content from FILE 3 above]
EOF
```

Step 5: Create Migration File

bash

```
cd ~/optiinfra/shared/database/migrations/versions

# Create migration file
cat > 003_workflow_history.py << 'EOF'
[Copy the entire migration content from FILE 4 above]
EOF

# Verify file created
ls -lh 003_workflow_history.py
# Expected: ~200 lines
```

Step 6: Test Imports

bash

```
cd ~/optiinfra

# Test if new models can be imported
python << 'EOF'
from shared.database.models.workflow_history import (
    WorkflowExecution,
    WorkflowStep,
    WorkflowArtifact,
    WorkflowType,
    WorkflowStatus,
    StepStatus,
    ArtifactType
)
print('✅ All workflow models import successfully')

# Test enums
print(f'✅ WorkflowType has {len(WorkflowType)} values')
print(f'✅ WorkflowStatus has {len(WorkflowStatus)} values')
print(f'✅ StepStatus has {len(StepStatus)} values')
print(f'✅ ArtifactType has {len(ArtifactType)} values')
EOF

# Expected output:
# ✅ All workflow models import successfully
# ✅ WorkflowType has 8 values
# ✅ WorkflowStatus has 6 values
# ✅ StepStatus has 6 values
# ✅ ArtifactType has 8 values
```

Step 7: Run Alembic Migration

bash

```
cd ~/optiinfra/shared/database

# Check current migration state
alembic current
# Expected: 002_agent_state_tables (current)

# Run the new migration
alembic upgrade head

# Expected output:
# INFO  [alembic.runtime.migration] Running upgrade 002_agent_state_tables -> 003_workflow_history, Create workflo
```

Step 8: Verify Tables Created

bash

```
# Connect to PostgreSQL
psql postgresql://optiinfra:password@localhost:5432/optiinfra

# List all tables (should now be 13 tables + 1 alembic)
\dt

# Expected output (10 from previous + 3 new):
#  workflow_executions
#  workflow_steps
#  workflow_artifacts
#  agent_configs
#  agent_states
#  agent_capabilities
#  agent_metrics
#  customers
#  agents
#  events
#  recommendations
#  approvals
#  optimizations
#  alembic_version

# Count should be 14 total (13 + alembic_version)

# Describe workflow_executions table
\d workflow_executions

# Expected: Should show all columns, indexes, foreign keys

# Describe workflow_steps table
\d workflow_steps

# Expected: Should show all columns, foreign key to workflow_executions

# Describe workflow_artifacts table
\d workflow_artifacts

# Expected: Should show all columns, foreign keys to both executions and steps

# Check new enums were created
\dT+
```

```
# Expected: Should see workflow_type, workflow_status, step_status, artifact_type

# Check agent_states got new column
\d agent_states

# Expected: Should see current_workflow_id column with foreign key

# Exit
\q
```

Step 9: Verify Relationships Work

bash

```
cd ~/optiinfra

# Test relationships in Python
python << 'EOF'
from shared.database.models import Agent, Customer, WorkflowExecution
from shared.database.session import SessionLocal

db = SessionLocal()

# Get a test agent
agent = db.query(Agent).first()
print(f"✅ Agent: {agent.name}")
print(f"   - Has workflow_executions relationship: {hasattr(agent, 'workflow_executions')}")

# Get a test customer
customer = db.query(Customer).first()
print(f"✅ Customer: {customer.name}")
print(f"   - Has workflow_executions relationship: {hasattr(customer, 'workflow_executions')}")

# Check workflow execution model
from shared.database.models.workflow_history import WorkflowExecution
print(f"✅ WorkflowExecution model loaded")
print(f"   - Has agent relationship: {hasattr(WorkflowExecution, 'agent')}")
print(f"   - Has customer relationship: {hasattr(WorkflowExecution, 'customer')}")
print(f"   - Has steps relationship: {hasattr(WorkflowExecution, 'steps')}")
print(f"   - Has artifacts relationship: {hasattr(WorkflowExecution, 'artifacts')}")

db.close()
print("\n✅ All relationship verifications passed!")
EOF

# Expected output:
# ✅ Agent: Cost Optimizer
#    - Has workflow_executions relationship: True
# ✅ Customer: Acme Corp
#    - Has workflow_executions relationship: True
# ✅ WorkflowExecution model loaded
#    - Has agent relationship: True
#    - Has customer relationship: True
#    - Has steps relationship: True
#    - Has artifacts relationship: True
```

```
#
# ✅ All relationship verifications passed!
```

🎯 PART 1 SUMMARY

Files Created/Modified:

✅ **shared/database/models/workflow_history.py** - 3 new models + 4 enums (~400 lines)
✅ **shared/database/models/core.py** - Modified (added 2 relationships)
✅ **shared/database/models/agent_state.py** - Modified (added 1 field + 1 relationship)
✅ **shared/database/models/init.py** - Modified (exports)
✅ **shared/database/migrations/versions/003_workflow_history.py** - Migration (~200 lines)

What Works Now:

✅ Can import all new models in Python
✅ Can run Alembic migration
✅ 3 new tables created in PostgreSQL
✅ All foreign keys working (cascades configured)
✅ All indexes created (15+ indexes)
✅ Agent-workflow relationship working
✅ Customer-workflow relationship working
✅ AgentState can track current workflow

Database Status:

```
Total Tables: 13 (100% PostgreSQL schema complete!)
├── Core (0.2a): 6 tables ✅
├── Agent State (0.2b): 4 tables ✅
└── Workflow History (0.2c): 3 tables ✅

Total Enums: 15
Total Indexes: 60+
Total Relationships: 25+
```

What's in Part 2:

- Seed data (workflow executions, steps, artifacts for test scenarios)
- Test fixtures (sample workflows, steps, artifacts)
- Test cases (18+ comprehensive tests)

- Integration tests (workflow execution flow, cascade deletes)
- All validation commands
- Troubleshooting guide
- Success criteria checklist
- Git commit instructions

⏭️ NEXT STEP

Download PART 2 to get:

- Complete seed data implementation (realistic workflow examples)
- Full test suite (18+ tests covering all scenarios)
- Step-by-step validation
- Troubleshooting for common issues
- Final checklist

File name: `FOUNDATION-0.2c-Workflow-History-PART2-Testing.md`

📥 HOW TO DOWNLOAD THIS FILE

1. Click the **"Copy"** dropdown button at the top of this artifact
2. Select **"Download as txt"**
3. Save as: `FOUNDATION-0.2c-Workflow-History-PART1-Code.md`

✅ VERIFICATION CHECKLIST (Part 1)

Before moving to Part 2, verify:

- [ ] `workflow_history.py` created with 3 models
- [ ] `core.py` updated with 2 new relationships
- [ ] `agent_state.py` updated with workflow link
- [ ] `__init__.py` exports all workflow models
- [ ] Migration file `003_workflow_history.py` created
- [ ] Migration runs successfully (`alembic upgrade head`)
- [ ] 3 new tables exist in PostgreSQL
- [ ] 4 new enum types created
- [ ] All indexes created
- [ ] Python imports work (no errors)
- [ ] Relationships work (verified in Python)

If all checked, you're ready for PART 2! ✅

✅ **PART 1 COMPLETE! Ready for PART 2?** 🚀